OpenGL Lecture 2

Geometry Drawing

Note: The material covered in this lecture is designated for the final lab

Review from last lecture

Create a new project then write the following in your C++ file:

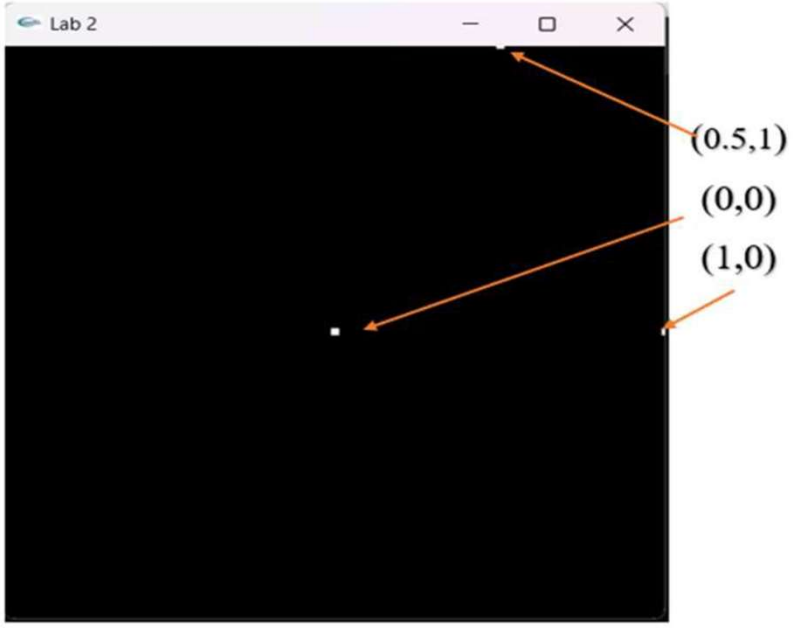
```
#include <Windows.h>
#include <GL/glut.h>
void draw() {
	glClear(GL_COLOR_BUFFER_BIT);


	// your drawing code here

	glFlush();
}
int main() {
	glutInitWindowSize(400, 400);
	glutCreateWindow("Lab 2");
	glutDisplayFunc(draw);
	glutMainLoop();
}
```

Now let's start drawing

- Write the following code in the drawing section specified below then run your code. Your output should be like:

```
glPointSize(5);
glBegin(GL_POINTS);
glVertex2d(0, 0);
glVertex2d(1, 0);
glVertex2d(0.5, 1);
glEnd();
```

always before the begin



Let's explain the code

In OpenGL we define vertices by their coordinates in the world coordinate system.

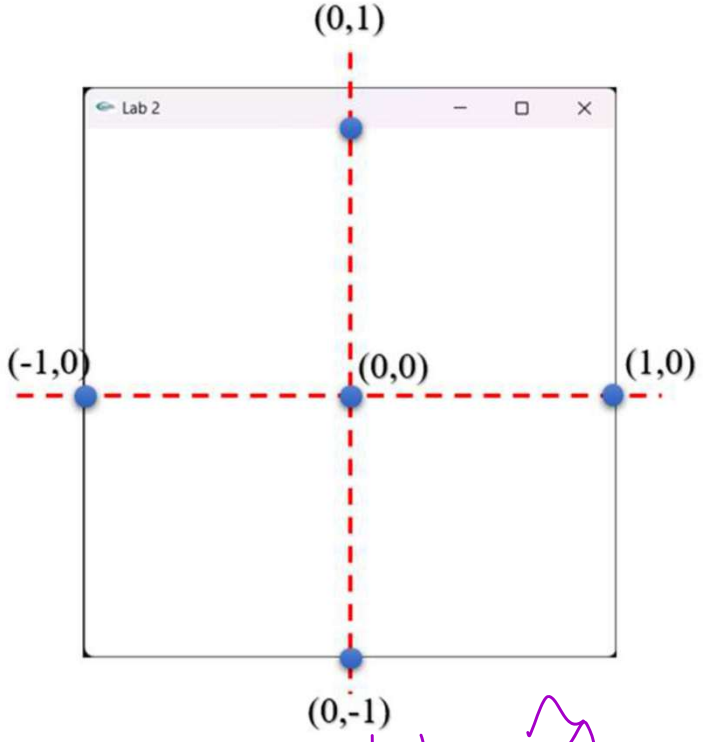
The default coordinate system is:

Xmin= -1

Xmax= 1

Ymin= -1

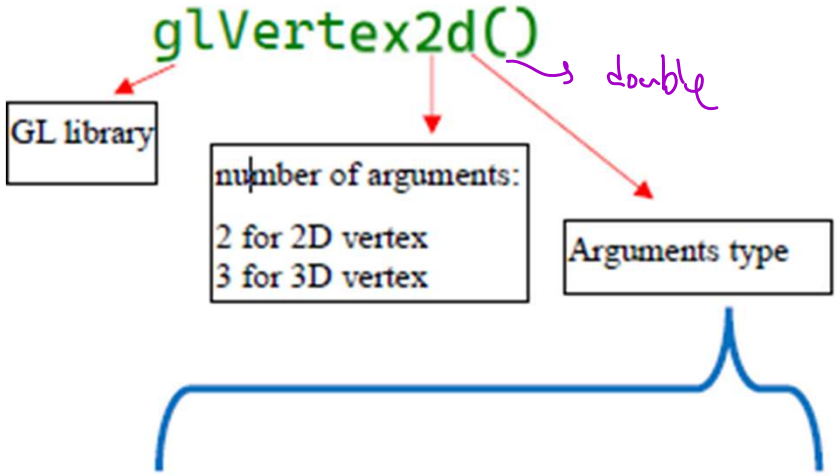
Ymax =1



Let's explain the code

To start drawing a 2D primitive in OpenGL :

- you must start with `glBegin(PRIMITIVE-TYPE)`
- then define all the vertices using `glVertex**().`
- Finally, we should tell that the primitive is complete by calling `glEnd();`



suffix	data type	C/C++ type
b	8-bit integer	signed char
s	16-bit integer	Short
i	32-bit integer	int or long
f	32-bit float	Float
d	64-bit float	Double
ub	8-bit unsigned number	unsigned char
us	16-bit unsigned number	unsigned short
ui	32-bit unsigned number	unsigned int or unsigned long

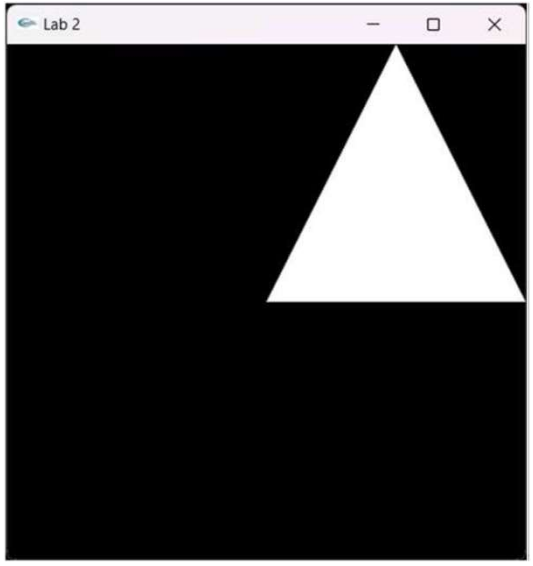
Excersise

- Draw more points with different sizes.
- Note: `glPointSize()` should be before `glBegin()`. So, to draw points with different sizes you must repeat the previous code for each group of points with the same size.
- Example:

```
glPointSize(4);
glBegin(GL_POINTS);
. . .
glEnd();

glPointSize(10);
glBegin(GL_POINTS);
. . .
glEnd();
```

Change the primitive

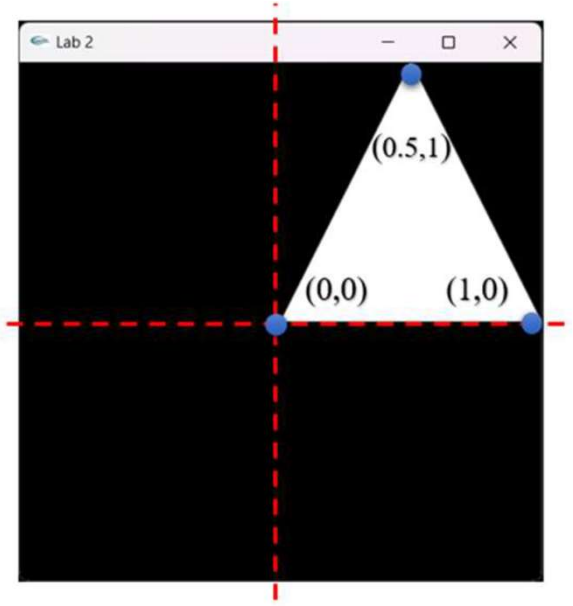
Update your previous code by replacing `GL_POINTS` with `GL_TRIANGLES`. Your output should be:



Explain the code

Update your previous code by replacing `GL_POINTS` with `GL_TRIANGLES`. Your output should be:

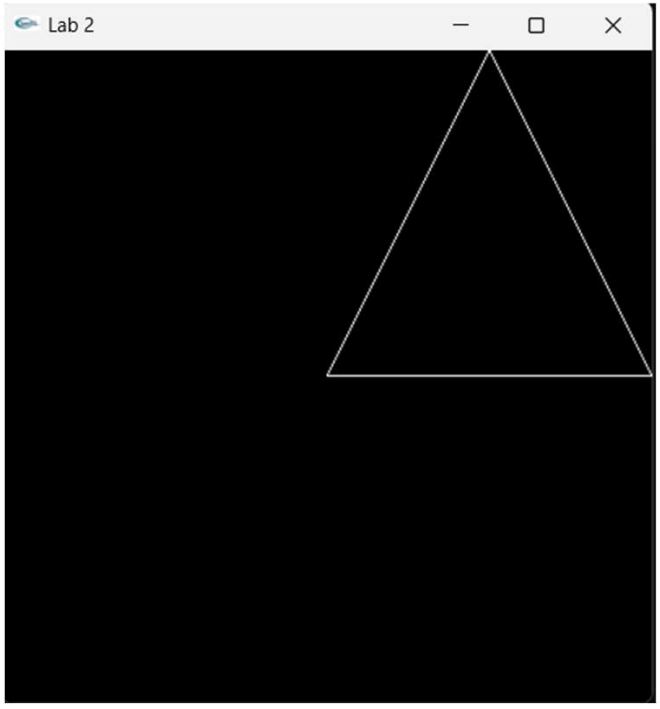
```
glPointSize(5);
glBegin(GL_TRIANGLES);
glVertex2d(0, 0);
glVertex2d(1, 0);
glVertex2d(0.5, 1);
glEnd();
```



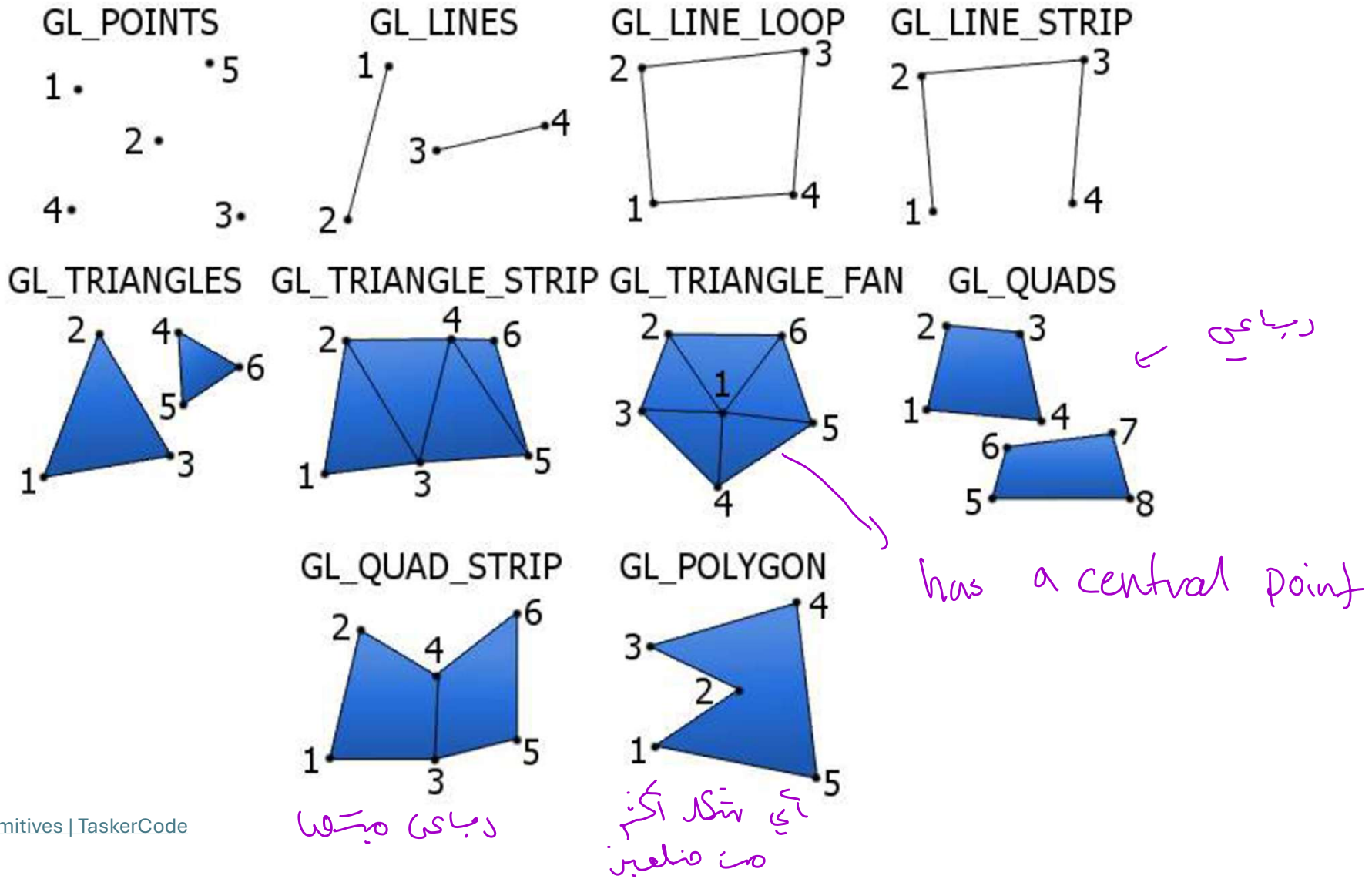
Another shape

Replace `GL_TRIANGLES` with `GL_LINE_LOOP` then run your code.

What is the difference?



Primitive	Description	Number of Vertices
GL_POINTS	Treats each vertex as a single point. Vertex *n* defines point *n*. *N* points are drawn.	Vertex *n* defines point *n*. *N* points are drawn.
GL_LINES	Treats each pair of vertices as an independent line segment.	Vertices *2n - 1* and *2n* define line *n*. *N/2* lines are drawn.
GL_LINE_STRIP	Draws a connected group of line segments from the first vertex to the last.	Vertices *n* and *n+1* define line *n*. *N - 1* lines are drawn.
GL_LINE_LOOP	Draws a connected group of line segments from the first vertex to the last, then back to the first.	Vertices *n* and *n + 1* define line *n*. The last line, however, is defined by vertices *N* and *1*. *N* lines are drawn.
GL_TRIANGLES	Treats each triplet of vertices as an independent triangle.	Vertices *3n - 2*, *3n - 1*, and *3n* define triangle *n*. *N/3* triangles are drawn.
GL_TRIANGLE_STRIP	Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices.	For odd *n*, vertices *n*, *n + 1*, and *n + 2* define triangle *n*. For even *n*, vertices *n + 1*, *n*, and *n + 2* define triangle *n*. *N - 2* triangles are drawn.
GL_TRIANGLE_FAN	Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices.	Vertices *1*, *n + 1*, *n + 2* define triangle *n*. *N - 2* triangles are drawn.
GL_QUADS	Treats each group of four vertices as an independent quadrilateral.	Vertices *4n - 3*, *4n - 2*, *4n - 1*, and *4n* define quadrilateral *n*. *N/4* quadrilaterals are drawn.
GL_QUAD_STRIP	Draws a connected group of quadrilaterals. One quadrilateral is defined for each pair of vertices presented after the first pair.	Vertices *2n - 1*, *2n*, *2n + 2*, and *2n + 1* define quadrilateral *n*. *N/2 - 1* quadrilaterals are drawn.
GL_POLYGON	Draws a single, convex polygon.	Vertices *1* through *N* define this polygon.



Source: Tutorial 2 – 2D Primitives | TaskerCode

Drawing Lines

In OpenGL, you can change the line width and style. Update your draw function as shown below then run your code :

```
void draw() {

    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    glVertex2d(-1, 0);
    glVertex2d(1, 0);
    glEnd();
    glFlush();

}
```

Now try to change the line width to be thicker using glLineWidth(10) before drawing the line. Then try different widths.

Line Style

Follow the following steps to change the line style.

1- First enable line stipple : `glEnable(GL_LINE_STIPPLE);`

2- Choose a line style : `glLineStipple(factor , pattern);`

3- Draw the line.

```
glClear(GL_COLOR_BUFFER_BIT);

glLineWidth(5.0);              // step 1: change line width
glEnable(GL_LINE_STIPPLE);     // step 2: enable line style
glLineStipple(1, 0X0CFF);      // step 3: select and apply line style
glBegin(GL_LINES);             // Step 4: draw the line
glVertex2d(-1, 0);
glVertex2d(1, 0);
glEnd();

glFlush();
```

any stylish prop is written befor begin like color or width or like slide down below

How to change line style

glLineStipple(factor , pattern);

The factor is a small integer number you can choose for example 1,2,3 etc...

Indicate the number in the Hexadecimal

0X00FF

00FF

0000000011111111

00000000111111110000000011111111000000001111111

Line Type	Factor	Pattern
— — —	1	0X00FF
— — —	2	0X00FF
- -	1	0XAAAA
— — —	3	0X0C0F
— — — —	4	0XAAAA

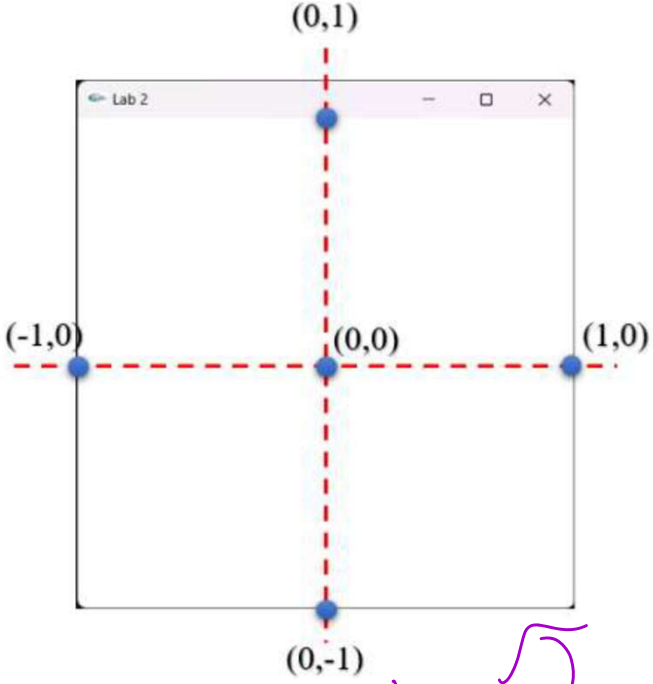
ما يحتاج أحفظهم ←
للاختبار وجه تكفي
وصف بيعطوني قيمته بالاختبار

Coordinate System

In the previous labs we were using the default coordinates as shown in the figure.

Where:

- Left = X min= -1
- Right= X max=1
- Bottom= Y min= -1
- Top= Y Max = 1



Coordinate System

We can change the coordinates using two functions `gluOrtho2D()` OR `glOrtho()`.

You can add these functions in the main() function at any line between `glutCreateWindow`() and `glutMainLoop()`

For 2D drawing :

gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);

For 2D and 3D we can use:

```
glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
Gldouble zNear, Gldouble zFar);
```

- ► zNear = front
- ► zFar= back (depth)

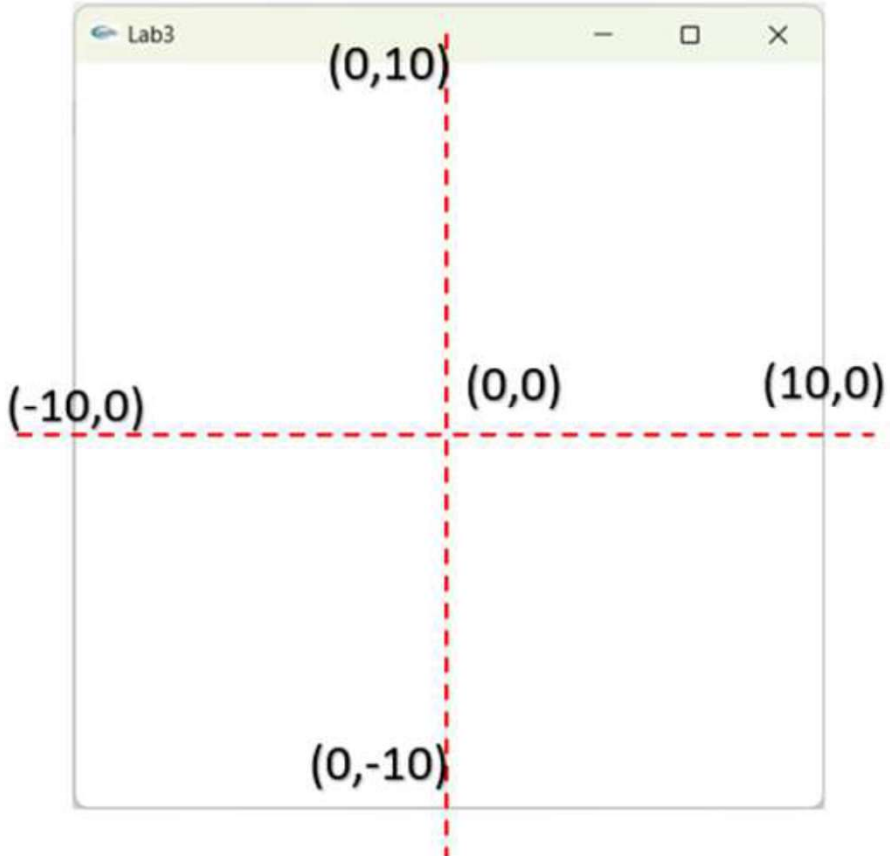
Note : to use glOrtho() for 2D drawing you can assign the values 0 and 1 to zNear and zFar respectively.

Excersise

Try different coordinates and see what happens:

- X min = -10
- X max =10
- Y min = -10
- Y Max = 10

```
gluOrtho2D(-10,10,-10,10)
```

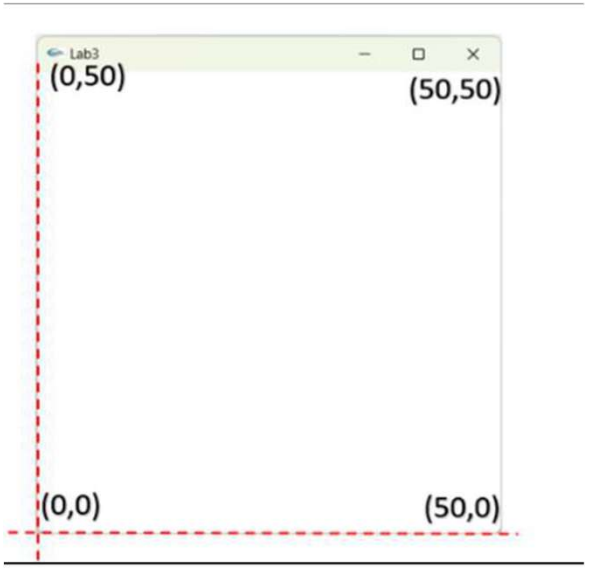


Excersise

Try different coordinates and see what happens:

- X min = 0
- X max =50
- Y min = 0
- Y Max = 50

```
gluOrtho2D(0,50,0,50)
```

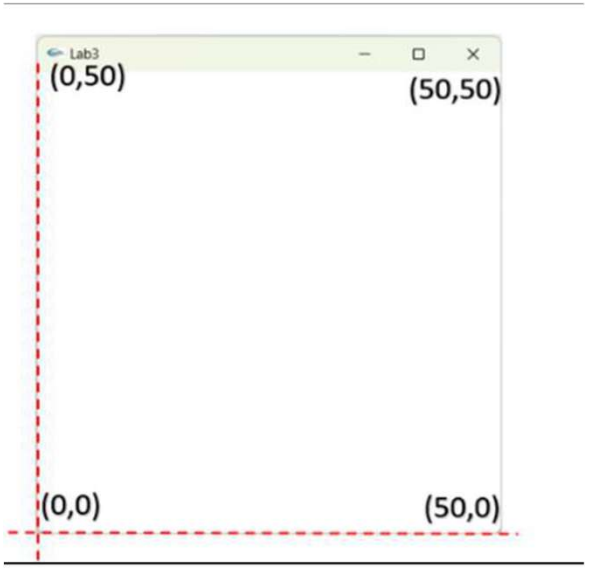


Excersise

Try different coordinates and see what happens:

- X min = 0
- X max =50
- Y min = 0
- Y Max = 50

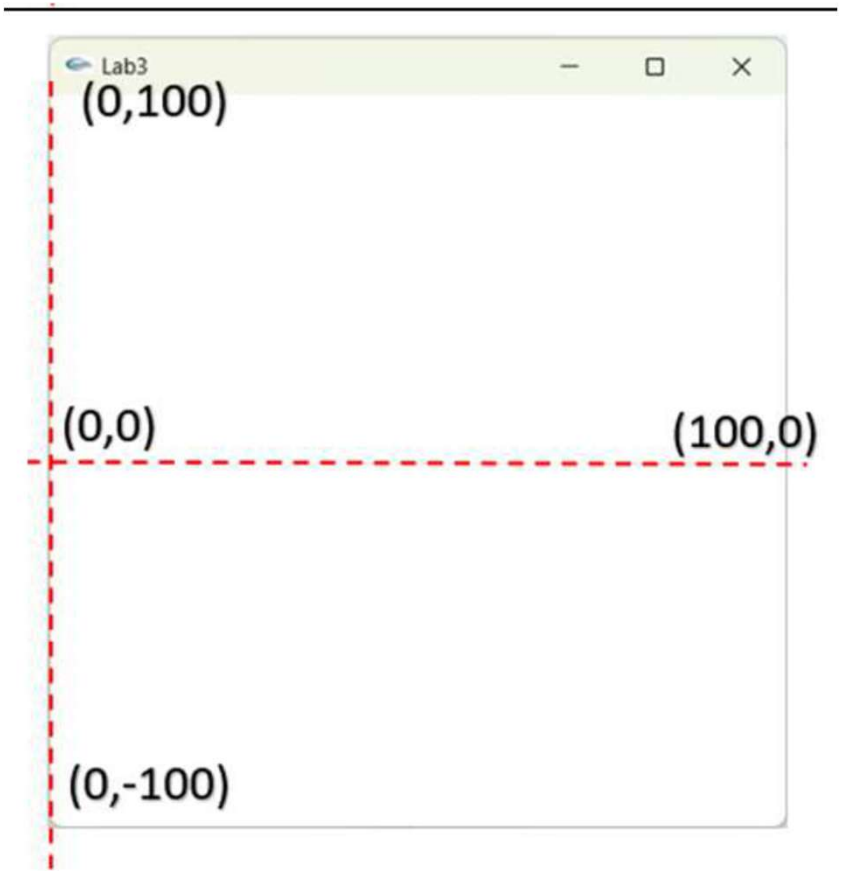
```
gluOrtho2D(0,50,0,50)
```



Excersise

- X min = 0
- X max =100
- Y min = -100
- Y Max = 100
- Z Near = 0
- Z Far = 1

glOrtho(0,100,-100,100,0,1)



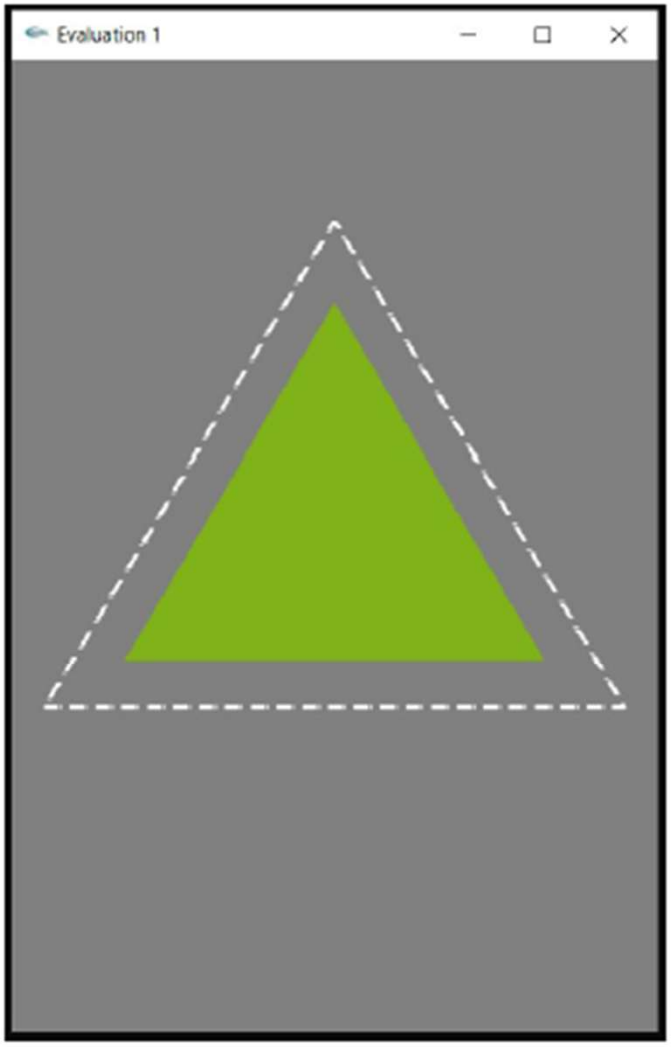
كزا تحبي الأسئلة

Exercise : 0

Write a C++ program to draw the figure shown in the window:

Where:

- Window Width : 400
- Window High : 600
- X min : -100
- X max : 100
- Y min : 0
- Y max : 300
- You can use any color
- You can use any Line style
- Note: main function is not given. However, you can use the provided cheat sheet



← drowing the line using line loop

تحبي بالاختبار ما يعتمد Solution أحضد

Cheat Sheet:

Functions		Constants	
glClear();	glutInitWindowSize();	GL_COLOR_BUFFER_BIT	GL_POINTS
glFlush();	glutInitWindowPosition();	GLUT_RGB	GL_LINES
glEnable();	glutInitDisplayMode();	GLUT_RGBA	GL_LINE_STRIP
glDisable();	glutCreateWindow();	GLUT_SINGLE	GL_LINE_LOOP
glPointSize();	glOrtho();	GLUT_DOUBLE	GL_TRIANGLES
glLineWidth();	gluOrtho2D();	GLUT_DEPTH	GL_TRIANGLE_STRIP
glLineStipple();	glClearColor();	GL_LINE_STIPPLE	GL_TRIANGLE_FAN
glColor3f();	glutDisplayFunc();		GL_QUADS
glColor3ub();	glutMainLoop();		GL_QUAD_STRIP
glVertex2i();	glTranslate*()		GL_POLYGON
glBegin();	glRotate*()		
glEnd();	glScale*()		